2009 Ninth International Conference on Intelligent Systems Design and Applications

Optimizing Linear and Quadratic Data Transformations for Classification Tasks

José M. Valls
Universidad Carlos III de Madrid
Madrid, Spain
jvalls@inf.uc3m.es

Ricardo Aler
Universidad Carlos III de Madrid
Madrid, Spain
aler@inf.uc3m.es

***Abstract*—Many classification algorithms use the concept of distance or similarity between patterns. Previous work has shown that it is advantageous to optimize general Euclidean distances (GED). In this paper, we optimize data transformations, which is equivalent to searching for GEDs, but can be applied to any learning algorithm, even if it does not use distances explicitly. Two optimization techniques have been used: a simple Local Search (LS) and the Covariance Matrix Adaptation Evolution Strategy (CMA-ES). CMA-ES is an advanced evolutionary method for optimization in difficult continuous domains. Both diagonal and complete matrices have been considered. The method has also been extended to a quadratic non-linear transformation. Results show that in general, the transformation methods described here either outperform or match the classifier working on the original data.**

***Keywords*-Data transformations; General Euclidean Distances; Evolutionary Computation; Evolutionary-based Machine Learning;**

Many classification algorithms use the concept of distance or similarity between patterns. This is specially true for local classification methods such as Radial Basis Neural Networks [1] or Nearest Neighbor classifiers [2]. It has been acknowledged that the Euclidean distance is not always the most appropriate for a particular domain, and for that reason other distances, like Mahalanobis or Chebyshev, have been proposed. For instance, the Mahalanobis distance normalizes attributes by taking into account variances and removing correlations. This distance is computed according to Eq. 1.

$$\begin{aligned} d_{ij} &= [(\mathbf{x_i}-\mathbf{x_j})^T\mathbf{S}^{-1}(\mathbf{x_i}-\mathbf{x_j})]^{1/2} \\ &= [(\mathbf{x_i}-\mathbf{x_j})^T\mathbf{M}^T\mathbf{M}(\mathbf{x_i}-\mathbf{x_j})]^{1/2} \end{aligned} \quad (1)$$

where d_{ij} is the Mahalanobis distance between vectors $\mathbf{x}_i$ and $\mathbf{x}_j$, S is the variance-covariance matrix of all vectors in the data set and $\mathbf{M}$ is the so-called Mahalanobis matrix [3], [4], [5]. However, the Mahalanobis distance is computed in an unsupervised way, without taking into account the class or the training error of the classifier. In other words, the distance is not optimized for the learning algorithm or the classification task.

In [9], Genetic Algorithms have been used to evolve Generalized Euclidean Distances (GED) for Radial Basis Neural Networks. GEDs look like Eq. 1, except that matrix $\mathbf{S}^{-1}$ (or matrix $\mathbf{M}$) is not computed from the data, but optimized by the Genetic Algorithm, where the fitness function is the classifier error on a training dataset.

However, this approach can only be used in classification algorithms that explicitly use distances, like RBNN, but not others like C4.5. However, a little algebra (Eq. 2) shows that the GED is equivalent to computing an Euclidean distance on a projected dataset, where the new patterns in the projected space are linear transformations of the original data: $\mathbf{x}' = \mathbf{M}\mathbf{x}$.

$$d_{ij} = [(\mathbf{M}\mathbf{x_i}-\mathbf{M}\mathbf{x_j})^T(\mathbf{M}\mathbf{x_i}-\mathbf{M}\mathbf{x_j})]^{1/2} \quad (2)$$

Therefore, in this paper we will use a search method to look for a transformation $\mathbf{M}$ (instead of a GED), that optimizes the training error of a learning algorithm. Any learning algorithm can be used but in the present work we will consider a nearest neighbor algorithm (KNN). Eq. 3 displays several examples of such transformations: (a) rotation of data points by θ degrees, (b) scaling of dimensions, and (c) projection onto the y coordinate. However, our method can be easily extended to some non-linear transformations. In this paper, we will report some results of a quadratic transformation.

$$\begin{aligned} (a)\ M &= \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix} \\ (b)\ M &= \begin{pmatrix} k & 0 \\ 0 & \frac{1}{k} \end{pmatrix} \\ (c)\ M &= \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \end{aligned}$$

Two optimization techniques will be applied. First, a simple local search method is proposed as a base line to compare with more advanced optimization methods. Second, we will use CMA-ES (Covariance Matrix Adaptation Evolution Strategy), one of the best evolutionary techniques for continuous optimization in difficult non-linear domains [8], [7]. CMA-ES is an Evolution Strategy where search is guided by a covariance matrix, which is adjusted and updated during the search process. CMA-ES works well in both local and global optimization tasks. One of the most interesting features of CMA-ES for our purposes is the self-adaptation of mutation (the update of the covariance matrix),

978-0-7695-3872-3/09 $26.00 © 2009 IEEE
DOI 10.1109/ISDA.2009.222

allowing for a finely tuned local search at the end of the optimization process.

The structure of this article is as follows. Section I describes the method, Section II describes the synthetic and real domains that have been used to test the approach, and also reports the results of the experiments. Finally, Section III draws some conclusions and points to future work.

I. Description of the method

In this paper we have applied two optimization algorithms for finding a linear transformation of a dataset that minimizes the classification error of a learning technique. The first one is a simple local search method that is used only as a baseline to compare with a second more advanced method: CMA-ES. In both methods, transformations are represented as matrices, which are coded directly as lists of real numbers. CMA-ES is extensively described in [8], [7] [1].

With respect to the local search method, Algorithm 1 provides a summary of the algorithm. First, matrix M is initialized to the identity matrix I (line 1). In fact this means that the starting matrix M corresponds to the Euclidean distance (Matrix I). Then, the training dataset T is transformed by matrix M (line 2) and the error E_M of the learning algorithm L on T_M is computed (line 3). In order to prevent overfitting, we have computed E_M by means of 10-fold crossvalidation. Then, the main loop is entered (line 4) where M is mutated (line 5) and the error of M' is computed (line 6). If M' is more accurate than M, then M' is kept (line 8).

Algorithm 1: Local Search

```
1  M = I
2  T_M = M * T
3  E_M = error(L, T_M)
4  while not stopping condition do
5      M' = mutate(M)
6      E_M' = error(L, T_M')
7      if E_M' <= E_M then
8          M ← M'
9          E_M = E_M'
10     end
11 end
```

We have considered two types of matrices: diagonal matrices (where all elements outside the diagonal are zero) and arbitrary non-diagonal matrices ($n \times n$ square matrices) that we call complete matrices. Using diagonal matrices amounts to just a weighting of the attributes by the corresponding element in the diagonal. The reason for testing these two types of structures is to check whether complete matrices are actually useful beyond a mere attribute weighting. In other words, complete matrices involve fitting many parameters ($n \times n$) and it is important to know whether they improve accuracy or rather produce overfitting because of the extra degrees of freedom, or underfitting because of not being able to adjust properly all the parameters.

Mutation in local search is carried out by adding a random sample from a Gaussian $N(0,1)$ distribution. We have set the probability of mutating a matrix element to $p_m = \frac{1}{\sqrt{n \times n}}$, where n is the number of attributes and $n \times n$ is the size (the number of elements) of the complete square matrix. This means that for a complete matrix, the average number of elements that will be mutated is n. This setting worked well in preliminary experiments. One of the main differences of the simple local search algorithm and CMA-ES is that in the latter, the mutation step is controlled by a multivariate Gaussian distribution where the covariance matrix is adjusted during the course of the search, allowing for a finely tuned optimization at the end of the optimization process. In the experimental section, we will determine to what extent this feature contributes to a high accuracy in the classification task.

With respect to CMA-ES, we have used the same parameter setting as in the local search. First, CMA-ES also starts from the identity matrix. Second, its initial step size is 1.0 (although CMA-ES will adjust this value in the course of the search). Third, both CMA-ES and local search stop after the same maximum number of matrix evaluations, which is determined experimentally for every domain.

Any learning algorithm L could be used. In this paper, the neighborhood-based algorithm KNN (with $k = 1$) has been selected. KNN makes very intuitive to think about what data transformations could be useful for this method. This has provided some guidance for proposing several of the synthetic domains that will be presented in the next section.

Although the main thrust of this paper is to optimize linear transformations (**x' = Mx**) for non-linear classification algorithms, the method can be easily extended to generate non-linear transformations by applying a non-linear function *F* to each component of the output, as shown in Eq. 3 for a 2-attribute problem.

$$\begin{pmatrix} x'_1 \\ x'_2 \end{pmatrix} = F\left(\begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) = \begin{pmatrix} F(m_{11}x_1 + m_{12}x_2) \\ F(m_{21}x_1 + m_{22}x_2) \end{pmatrix} \quad (3)$$

In the case of $F(x) = x^n$, the method generates attributes made of polynomials of degree n. Such polynomials have been typically used for providing non-linearity to linear methods [11]. For simplicity, let us suppose that $F(x) = x^2$, $A = 2$, and that the original dataset $\{(x_1, x_2)\}$ is slightly transformed to $\{(x_1, x_2, 1)\}$. Also, let us suppose that 2×3 matrices are to be evolved. In that case, Eq. 4 shows that the transformed attributes (x'_1, x'_2) become quadratic expressions (Eq. 4). Thus, a linear separation in the transformed

[1] We have used the Matlab code available at `http://www.lri.fr/~hansen/cmaes_inmatlab.html`

space (x'_1, x'_2) would be equivalent to a quadratic separation in the original space $\{(x_1, x_2)\}$. This method permits to work with polynomials of degree 2 (n in general), without explicitly considering the monomials x_i^2 or $x_i x_j$.

$$\begin{aligned}\begin{pmatrix} x'_1 \\ x'_2 \end{pmatrix} &= F_{x^2}\left(\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}\right) \\ &= \begin{pmatrix} (m_{11}x_1 + m_{12}x_2 + m_{13})^2 \\ (m_{21}x_1 + m_{22}x_2 + m_{23})^2 \end{pmatrix} \\ &= \begin{pmatrix} \sum_i A_{1i}x_i + \sum_{i\neq j} B_{1ij}x_i x_j + C_1 \\ \sum_i A_{2i}x_i + \sum_{i\neq j} B_{2ij}x_i x_j + C_2 \end{pmatrix} \quad (4)\end{aligned}$$

To summarize, our method can either search for linear transformations for non-linear learning algorithms (like KNN), or search for non-linear transformations (e.g. quadratic) for linear/non-linear classifiers. For the latter case, we have chosen a linear classifier: Fisher Discriminant Analysis.

II. Experiments

In this Section, we will carry out experiments on several domains where linear and non-linear transformations are explored.

A. Domains

We have used five synthetic domains and three real world domains. All of them correspond to classification problems and have numerical attributes. They are described next.

1) Artificial data domains: We have used a well-known synthetic dataset, the **Ripley** [6] data domain, which has been widely used in Machine Learning literature, and three more artificial domains that we have called **RandomAttr**, **Straight0** and **Straight45**, specifically designed to check if approach works properly. We have also generated the **Quadratic** domain with the aim of exploring quadratic transformations.

In the **Ripley** data set each pattern has two real-valued coordinates and a class which can be 0 or 1. Each class corresponds to a bimodal distribution that is a balanced composition of two normal distributions. Covariance matrices are identical for all the distributions and the centers are different. One of the issues that make this domain interesting is the big overlap existing between both classes.

RandomAttr is a two-class domain with four real-valued attributes x_1, x_2, x_3, x_4. The examples have been randomly generated following an uniform distribution in the interval $[0, 1]$ for attributes x_1, x_2 and the interval $[0, 100]$ for attributes x_3, x_4. If $x_1 < x_2$ then the example is labeled as class '1'. Otherwise, if $x_1 > x_2$ the example belongs to class '0'. Thus, attributes x_3 and x_4 are irrelevant. Because the ranges of irrelevant attributes are much bigger, the classification accuracy of KNN is very bad (about 50%). The dataset is composed of 300 examples, 150 from each class.

Straight-45 is a two-class domain with two real-valued attributes. The examples have been generated in this way: initially 100 examples of class 1 are located ar regular intervals in a straight line passing through the origin $(0, 0)$ with an angle of 45 degrees respect to the horizontal axe. The distance between two consecutive points is 1. 100 examples of class 0 are generated in the same way in a parallel straight line passing through the point $(0, -1)$ in such a way that the nearest point of a given point always belongs to the opposite class, because it is located in the opposite parallel straight line. Then all points are perturbed by adding to each coordinate a random number uniformly distributed in $[-0.5, 0.5]$.

The idea behind this domain is that the nearest neighbor algorithm will achieve a very bad result because most of the times the nearest neighbor of a given point belongs to the opposite class. But certain transformations of the data involving rotations and coordinate scaling will allow a good classification rate. Figure 1 (left) shows a graphical representation of a subset of the domain.

Straight-0 is very similar to **Straight45**. The only difference is that all the points have been rotated 45 degrees anti-clockwise. The motivation for using this domain is that in this case with a simpler transformation the data could be properly classified because no rotation is needed. In Figure 1 (rigth) we can see the representation of a subset of points. In short, **Straight45** requires both rotation and scaling (a complete matrix) whereas **Straight-0** requires scaling only (a diagonal matrix).

Quadratic Domain. This domain was generated in the following way. First, coefficients were randomly generated for the quadratic expression 5. There are 8 variables x_i, where i ranges from 1 to 8. Then, 2000 data points were generated by giving random values in the range -2 to 2, to variables x_i. Finally, the data points were assigned the positive class if expression 5 is larger than zero, and the negative class otherwise. The goal of the learning algorithm is to re-discover the quadratic frontier represented by Eq. 5.

$$\sum_i A_i x_i^2 + \sum_i B_i x_i + \sum_{ij} C_{ij} x_i x_j + D = 0 \quad (5)$$

2) Real world data domains: We have used the well known **Iris**, **Car**, **Bupa**[2], and **Wine** domains from the UCI Machine Learning Repository [3]. Table I displays the characteristics of these UCI domains.

[2]Liver Disorders Data Set
[3]http://archive.ics.uci.edu/ml/

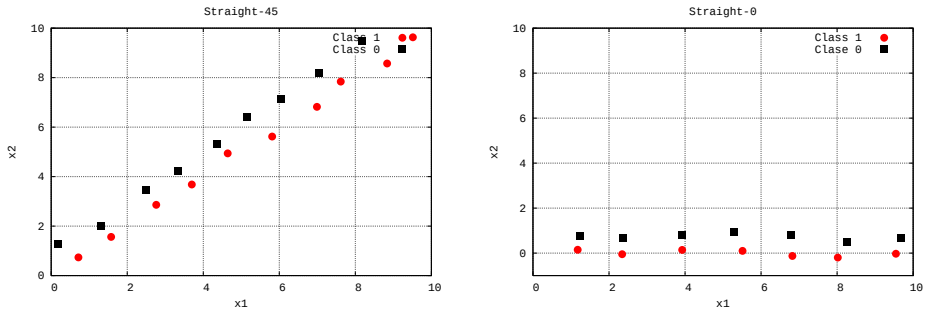

Figure 1. Subsets of the Straight-45 and Straight-0 domains

B. Experimental Results. Evolving linear transformations for KNN

In this subsection we show the experimental results obtained by our method when linear transformations are evolved and KNN (with k=1) is used as classifier in the transformed space. As we explained in Section I, two optimization techniques have been used: a basic Local Search method (LS) and CMA-ES.

The parameters for the search methods have been chosen in the following way: for CMA-ES, the initial standard deviation has been set to 1.0 for all the experiments. For LS, the standard deviation for the Gaussian mutation of an element is also 1.0 in all cases. The maximum number of fitness evaluations has been set to the same value for both search methods by means of preliminary experiments for every domain. It is considered that the training error rate has converged when it does not decrease by more than 10^{-3} in two successive iterations. For LS, the probability of mutating a matrix element is $p_m = \frac{1}{\sqrt{n \times n}}$, as explained in Section I. This probability is the same for all the experiments. The rest of CMA-ES parameters are set to their default values.

In all domains we compare the classification results in five situations: for each fold, KNN classifies the original data, the data transformed by a diagonal matrix optimized by CMA-ES, by a diagonal matrix optimized by LS, by a complete matrix optimized by CMA-ES and finally, by a complete matrix optimized by LS. In all cases linear transformations are done by means of square matrices and thus, the dimension of the transformed spaces remain unaltered. Using a diagonal matrix is equivalent to scaling the original data coordinates. If a complete matrix is used there are no restrictions, being the linear transformation completely general.

Table I
UCI DOMAINS CHARACTERISTICS

Domain	Instances	Attributes	Classes
Iris	150	4	3
Car	1728	6	4
Bupa	345	7	2
Wine	178	13	3

Table II shows the mean classification accuracy rates obtained for all the domains and their standard deviations. The means have been obtained by averaging 10 executions of a 10-fold crossvalidation procedure. The best results have been highlighted in boldface. Differences between each one of the four methods and the original data have been marked with a † if they are statistically significant ($\alpha = 0.05$). A Corrected Repeated 10x10-fold Crossvalidation T-Test has been used in all cases for testing [10].

In the **Straight0** domain, with a simple transformation of the space, just compressing the x_1 coordinate, a good classification rate can be achieved because points belonging to the same class can get as close to each other as needed. This simple transformation can be done with a diagonal matrix and therefore with a complete matrix too. The results show that KNN only obtains a classification rate of 10.5% on the original data set, but if the data is linearly transformed with a diagonal matrix the rate is 100% and 99.7% if a complete matrix is used. In this case both optimization methods achieve the same results for each matrix type. The slightly worse result obtained with the complete matrix is explained because the number of parameters to adjust is bigger.

In the **Straight45** domain, we see that a more complex transformation must be done because it is not enough to scale the coordinates but to rotate the points as well. This linear transformation cannot be obtained with a diagonal matrix. The second row of Table II shows the results as expected: KNN obtains a very bad classification accuracy on the original data (8.45%). A diagonal matrix is useless to obtain an adequate transformation and we can see that the results are very bad too, around 8%, independently of the optimization method used. On the contrary, when a complete matrix is used, the results are near to 100%.

The **RandomAttr** domain has two irrelevant attributes whose numeric range is much bigger that the relevant attributes range. That is the reason why the accuracy of KNN on the original data is 50%. Scaling the irrelevant attributes should be enough to attain a much better accuracy, thus both diagonal and complete matrices should be appropriate. The

Table II
CLASSIFICATION RATE (PERCENTAGE) WITH KNN (K=1) FOR THE ORIGINAL DATA SET AND FOR THE TRANSFORMED DATA WHEN DIAGONAL AND COMPLETE MATRICES ARE USED.

	Original Data	CMA-ES Diagonal	LS Diagonal	CMA-ES Complete	LS Complete
Straight-0	10.50 ± 1.99	$\mathbf{100.00} \pm 0$†	$\mathbf{100.00} \pm 0$†	99.70 ± 0.35†	99.80 ± 0.35†
Straight-45	8.45 ± 2.07	8.55 ± 2.05	8.15 ± 2.42	$\mathbf{98.70} \pm 0.54$†	98.55 ± 0.50†
RandomAtt	50.03 ± 1.48	$\mathbf{95.33} \pm 2.21$†	92.93 ± 1.55†	81.13 ± 4.62†	59.77 ± 3.41
Ripley	88.60 ± 0.49	$\mathbf{88.84} \pm 0.35$	88.77 ± 0.52	88.43 ± 0.54	88.57 ± 0.38
Car	87.47 ± 0.15	95.82 ± 0.30†	95.75 ± 0.25†	$\mathbf{97.39} \pm 0.25$†	97.20 ± 0.39†
Iris	95.87 ± 0.28	94.67 ± 0.99	94.33 ± 1.01	$\mathbf{96.07} \pm 0.80$	95.87 ± 1.03
Bupa	62.20 ± 1.29	60.52 ± 1.29	58.58 ± 1.78	$\mathbf{65.33} \pm 2.39$	63.54 ± 3.01
Wine	76.38 ± 1.42	93.63 ± 0.9†	$\mathbf{94.61} \pm 1.53$†	85.61 ± 2.33†	77.62 ± 1.48

results show that diagonal matrices obtain results over 90%. It can also be seen that in this domain, complete matrices do not perform as well as diagonal matrices because the number of elements to adjust is bigger and only a scaling of the coordinates is necessary.

With regard to the remaining domains, only the statistically significant results will be described. For instance, Bupa and Iris display some improvement, but it is not significant. In the case of Car, all methods display large and significant differences with respect to the original data. The best one is complete-CMA with 97.39% (vs. 87.47% of the original data). Additional significance tests show that differences between diagonal-CMA and complete-CMA are significant but not large: 95.82% vs. 97.39% (similar results can be seen for LS). Contrariwise, in the Wine domain, it is diagonal matrices that obtain the best results (94.61% diagonal-LS vs. 76.38% original data). Complete matrices perform poorly and the differences with diagonal matrices are large and significant (a similar behavior was observed in the RandomAttr domain).

Summarizing the results, it can be observed that the four methods behave as expected in the artificial domains, although in one of them (RandomAttr) complete matrices display worse accuracies than diagonal ones, even though the set of complete matrices include diagonal ones. A similar behavior can be observed in the Wine domain. Presumably this is due to complete matrices having more parameters to be adjusted. Second, in two of the real domains (Car and Wine), the methods significantly improve the original data, while in the other two (Iris and Bupa) there is no significant improvement, but no significant worsening either. Third, in at least two cases (Straight45 and Car) complete matrices are necessary to obtain the best results (significant differences[4]), although in the rest of domains, diagonal matrices seem to be enough (there are no significant differences between diagonal and complete matrices in Ripley, Iris, and Bupa, while diagonal matrices are significantly better in Wine and RandomAttr). Fourth, other tests show that there are almost no significant differences in any of the real domains, between CMA-ES and LS (except in Wine, where Complete CMA-ES significantly outperforms Complete LS).

[4]Please, note that these tests are additional significance tests: they do not appear in Table II in order to avoid cluttering the table.

C. Experimental Results. Evolving quadratic transformations for a linear classifier

As it was explained in section I, our approach can also search for non-linear transformations (e.g. quadratic) for linear classifiers. We have done the experiments using a discriminant analysis based classifier applied to the Quadratic, Ripley, Car, Iris, Bupa, and Wine domains. As in the former subsection, the classifier has been applied to either the original data and the transformed data by means of quadratic transformations. In this case, only complete matrices are used. As before, both CMA-ES and LS have been used to optimize the projection matrices. Table III shows the mean classification accuracies and Standards deviations obtained by the linear classifier on either the original and the projected data. As in the previous section, a repeated 10x10 crossvalidation has been carried out (and the Corrected Repeated 10x10-fold Crossvalidation T-Test has been used for statistical significance [10].). Column 2 refers to the original data, column 3 and 4 refers to the transformed data when the matrices have been optimized by CMA-ES and LS.

Table III
CLASSIFICATION RATE WITH LINEAR CLASSIFIER FOR THE ORIGINAL SET AND THE PROJECTED DATA WITH QUADRATIC PROJECTIONS.

	Original Data	CMA-ES Complete	LS Complete
Quadratic	64.48 ± 0.20	$\mathbf{93.67 \pm 0.26}$†	87.00 ± 1.19†
Ripley	87.86 ± 0.11	89.27 ± 0.26†	$\mathbf{89.30} \pm 0.38$†
Car	76.06 ± 0.24	$\mathbf{88.24 \pm 0.47}$†	86.30 ± 0.37†
Iris	$\mathbf{98.00 \pm 0}$	96.67 ± 0.66	96.74 ± 0.70
Bupa	62.32 ± 8.21	$\mathbf{68.73 \pm 1.44}$†	66.92 ± 2.17
Wine	$\mathbf{98.69} \pm 0.39$	97.94 ± 1.01	98.31 ± 0.84

In the Quadratic domain, the accuracy on the transformed data is much better than the obtained on the original data,

as expected. With respect to the rest of domains, CMA-ES outperforms significantly the original data in Ripley, Car, and Bupa but not in Iris and Wine. Differences between CMA-ES and LS are significant for the Quadratic domain and Car, but not for the rest.

III. Conclusion

In this work we have applied two optimization algorithms (Local Search and CMA-ES) for finding linear and quadratic transformations for datasets (represented by matrices), in order to improve the accuracy on classification tasks. The goal of the search is to find matrices that minimize the classification error on the training data. We have considered both diagonal and complete matrices. The non-linear classifier KNN has been used for the linear transformations, and the linear Fisher Discriminant for the quadratic ones. The method has been tested with nine different domains, both synthetic and real, and the results show that in general matrices found by the methods described, either outperform or match the classifier on the untransformed data. CMA-ES tends to get better results than the simpler LS method, although in many cases differences are not significant. In some of the domains, complete matrices are required in order to obtain the best results, but in others, complete matrices can get stuck in local minima. The latter issue is an important issue that will be dealt with in future research.

In this paper, we have used KNN and a linear classifier as the base algorithms, but because we use domain independent optimization methods, the approach can be applied to any other classifier. In the future, we will test what other learning methods benefit from data transformations. Also, using rectangular matrices instead of square ones, we could study the performance of the method as a dimensionality reduction technique in a similar vein to [12]. Finally, non-standard CMA-ES features, like re-starts, might be of use to solve the local minima / underfitting issues for complete matrices reported in this paper.

Acknowledgment

This work has been funded by the Spanish Ministry of Science under contract TIN2008-06491-C04-03 (MSTAR project)

References

[1] J.E. Moody and C. Darken. Fast learning in networks of locally tuned processing units. *Neural Computation*, 1:281–294, 1989.

[2] T.M. Cover and P.E. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inform. Theory*, 13(1):21–27,1967.

[3] C.G.Atkenson, A.W.Moore, and S.Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11:11–73, 1997.

[4] J. T. Tou and R. C. Gonzalez. *Pattern Recognition Principles*. Addison-Wesley, 1974.

[5] S. Weisberg. *Applied Linear Regression*. New York: John Wiley and Sons, 1985.

[6] B.D. Ripley. Pattern Recognition and Neural Networks *Cambridge: Cambridge University Press*, 1996.

[7] N. Hansen and A. Ostermeier. Completely Derandomized Self-adaptation in Evolution Strategies. *Evolutionary Computation* 9(2):159–195. 2001.

[8] A. Ostermeier, A. Gawelczyk and N. Hansen. A Derandomized Approach to Self-Adaptation of Evolution Strategies. *Evolutionary Computation*. 4(2):369-380. 1994.

[9] J.M. Valls, R. Aler and O. Fernández. Evolving Generalized Euclidean Distances for Training RBNN. *Computing and Informatics*. 26:33-43. 2007.

[10] R.R. Bouckaert and E. Frank. Evaluating the Replicability of Significance Tests for Comparing Learning Algorithms. *Advances in Knowledge Discovery and Data Mining (PAKDD)*. 3-12. 2004.

[11] A. Sierra. High-order Fishers discriminant analysis. *Pattern Recognition*, 35:1291-1302, 2002.

[12] A. Sierra and A. Echeverra. Evolutionary discriminant analysis. *IEEE Transactions on Evolutionary Computation*, vol. 10 (1), 81-92, 2006